Task 7.1 & 7.2 – ShapeDrawer – Save / Load

ExtensionMethods.cs

```
using System;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer
{
    public static class ExtensionMethods
    {
        public static int ReadInteger(this StreamReader reader)
        {
            return Convert.ToInt32(reader.ReadLine());
        }

        public static float ReadSingle(this StreamReader reader)
        {
            return Convert.ToSingle(reader.ReadLine());
        }

        public static Color ReadColor(this StreamReader reader)
        {
            return Color.RGBColor(reader.ReadSingle(),
reader.ReadSingle(), reader.ReadSingle());
        }

        public static void WriteColor(this StreamWriter writer, Color clr)
        {
            writer.WriteLine("{0}\n{1}\n{2}", clr.R, clr.G, clr.B);
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using SplashKitSDK;

namespace ShapeDrawer;

public class Program
{
    // Private enumeration for shape kinds
    private enum ShapeKind
    {
        Rectangle,
        Circle,
        Line
    }

    public static void Main()
    {
        Window window = new Window("Shape Drawer - Multiple Shape Kinds",
800, 600);

        // Create a new Drawing object
        Drawing myDrawing = new Drawing();

        // Variable to track which kind of shape to add
        ShapeKind kindToAdd = ShapeKind.Circle;

        // Number of parallel lines (last digit of student ID is 1, so
X=1)
        int parallelLines = 1;

        // Step 9: File path for saving/loading -
        string filePath =
"/Users/minthukyawkhaung/Desktop/TestDrawing.txt"; // Mac
        //string filePath =
        do
        {
            SplashKit.ProcessEvents();
            SplashKit.ClearScreen();
```

```
            // Check for R key to select Rectangle
            if (SplashKit.KeyTyped(KeyCode.RKey))
            {
                kindToAdd = ShapeKind.Rectangle;
            }

            // Check for C key to select Circle
            if (SplashKit.KeyTyped(KeyCode.CKey))
            {
                kindToAdd = ShapeKind.Circle;
            }

            // Check for L key to select Line
            if (SplashKit.KeyTyped(KeyCode.LKey))
            {
                kindToAdd = ShapeKind.Line;
            }

            // Step 9: Check for S key to save
            if (SplashKit.KeyTyped(KeyCode.SKey))
            {
                try
                {
                    myDrawing.Save(filePath);
                    Console.WriteLine("Drawing saved successfully to: " +
filePath);
                }
                catch (Exception e)
                {
                    Console.Error.WriteLine("Error saving file: {0}",
e.Message);
                }
            }

            // Step 16: Check for O key to open/load
            if (SplashKit.KeyTyped(KeyCode.OKey))
            {
                try
                {
                    myDrawing.Load(filePath);
                    Console.WriteLine("Drawing loaded successfully from: "
+ filePath);
```

```
                }
                catch (Exception e)
                {
                    Console.Error.WriteLine("Error loading file: {0}",
e.Message);
                }
            }

            // Check if left mouse button is clicked
            if (SplashKit.MouseClicked(MouseButton.LeftButton))
            {
                // Get mouse position
                float mouseX = SplashKit.MouseX();
                float mouseY = SplashKit.MouseY();

                // Declare myShape variable
                Shape? myShape = null;

                if (kindToAdd == ShapeKind.Rectangle)
                {
                    myShape = new MyRectangle();
                }
                else if (kindToAdd == ShapeKind.Circle)
                {
                    myShape = new MyCircle();
                }
                else  // Line
                {
                    // Draw parallelLines number of lines at the same time
                    for (int i = 0; i < parallelLines; i++)
                    {
                        Shape lineShape = new MyLine(Color.Red, mouseX,
mouseY + (i * 10), mouseX + 100, mouseY + (i * 10));
                        myDrawing.AddShape(lineShape);
                    }
                }

                // Add the shape to the drawing
                if (myShape != null)
                {
                    myShape.X = mouseX;
                    myShape.Y = mouseY;
```

```
                    myDrawing.AddShape(myShape);
                }
            }

            // Check if spacebar is pressed
            if (SplashKit.KeyTyped(KeyCode.SpaceKey))
            {
                // Change the background color to a new random color
                myDrawing.Background = SplashKit.RandomColor();
            }

            // Check if right mouse button is clicked
            if (SplashKit.MouseClicked(MouseButton.RightButton))
            {
                // Get current mouse position
                Point2D mousePos = SplashKit.MousePosition();
                // Tell myDrawing to SelectShapesAt the current mouse
pointer position
                myDrawing.SelectShapesAt(mousePos);
            }

            // Check if Delete key or Backspace key is pressed
            if (SplashKit.KeyTyped(KeyCode.DeleteKey) ||
SplashKit.KeyTyped(KeyCode.BackspaceKey))
            {
                // Get all selected shapes and remove them from the
drawing
                List<Shape> selectedShapes = myDrawing.SelectedShapes;
                foreach (Shape shape in selectedShapes)
                {
                    myDrawing.RemoveShape(shape);
                }
            }

            // Tell myDrawing to Draw
            myDrawing.Draw();

            SplashKit.RefreshScreen();

        } while (!window.CloseRequested);
    }
}
```

Drawing.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer
{

    public class Drawing
    {
        // Private fields
        private readonly List<Shape> _shapes;
        private Color _background;

        // Constructor
        public Drawing(Color background)
        {
            _shapes = new List<Shape>();
            _background = background;
        }

        // Default constructor using Color.White
        public Drawing() : this(Color.White)
        {
            // other steps could go here…
        }

        //Properties
        public List<Shape> SelectedShapes
        {
            get
            {
                List<Shape> result = new List<Shape>();
                foreach (Shape s in _shapes)
                {
                    if (s.Selected)
                    {
                        result.Add(s);
                    }
                }
```

```
            return result;
        }
    }

    public int ShapeCount
    {
        get { return _shapes.Count; }
    }

    public Color Background
    {
        get { return _background; }
        set { _background = value; }
    }

    //Methods
    public void Draw()
    {
        SplashKit.ClearScreen(_background);
        foreach (Shape s in _shapes)
        {
            s.Draw();
        }
    }

    // SelectShapesAt method that selects/deselects shapes at given
point
    public void SelectShapesAt(Point2D pt)
    {
        foreach (Shape s in _shapes)
        {
            if (s.IsAt(pt))
            {
                s.Selected = true;
            }
            else
            {
                s.Selected = false;
            }
        }
    }
```

```
public void AddShape(Shape s)
{
    _shapes.Add(s);
}

public void RemoveShape(Shape s)
{
    _shapes.Remove(s);
}

// Step 4,31: Save method to save drawing to file
public void Save(string filename)
{
    StreamWriter writer = new StreamWriter(filename);

    try
    {
        writer.WriteColor(Background);
        writer.WriteLine(ShapeCount);
        foreach (Shape s in _shapes)
        {
            s.SaveTo(writer);
        }
    }
    finally
    {
        writer.Close();
    }
}

// Step 1,30: Load method to load drawing from file
public void Load(string filename)
{
    StreamReader reader = new StreamReader(filename);
    string? kind;
    Shape? s;

    try
    {
        Background = reader.ReadColor();
        int count = reader.ReadInteger();
        _shapes.Clear();
```

```
                for (int i = 0; i < count; i++)
                {
                    kind = reader.ReadLine();
                    switch (kind)
                    {
                        case "Rectangle":
                            s = new MyRectangle();
                            break;
                        case "Circle":
                            s = new MyCircle();
                            break;
                        case "Line":
                            s = new MyLine();
                            break;
                        default:
                            throw new InvalidDataException("Unknown shape
kind: " + kind); // Step 28: Handle unknown shape kind
                    }

                    s.LoadFrom(reader);
                    AddShape(s);
                }
            }
            finally
            {
                reader.Close();
            }
        }
    }
}
```

Shape.cs

```
using System;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer;

public abstract class Shape
{
    // Fields
    private Color _color;
    private float _x;
    private float _y;
    private bool _selected;

    // Default constructor
    public Shape() : this(Color.Yellow)
    {
    }

    // Overloaded constructor that takes color as argument
    public Shape(Color color)
    {
        _color = color;
        _x = 0.0f;
        _y = 0.0f;
        _selected = false;
    }

    // Properties
    public Color Color
    {
        get { return _color; }
        set { _color = value; }
    }

    public float X
    {
        get { return _x; }
        set { _x = value; }
    }
```

```
    public float Y
    {
        get { return _y; }
        set { _y = value; }
    }

    public bool Selected
    {
        get { return _selected; }
        set { _selected = value; }
    }

    // Abstract methods - must be implemented by derived classes
    public abstract void Draw();
    public abstract void DrawOutline();
    public abstract bool IsAt(Point2D pt);

    // Step 5: Virtual SaveTo method - can be overridden by derived
classes
    public virtual void SaveTo(StreamWriter writer)
    {
        writer.WriteColor(Color);
        writer.WriteLine(X);
        writer.WriteLine(Y);
    }

    // Step 13: Virtual LoadFrom method - can be overridden by derived
classes
    public virtual void LoadFrom(StreamReader reader)
    {
        Color = reader.ReadColor();
        X = reader.ReadSingle();
        Y = reader.ReadSingle();
    }
}
```

MyRectangle.cs

```
using System;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer;

public class MyRectangle : Shape
{
    // Fields
    private int _width;
    private int _height;

    // Default constructor
    public MyRectangle() : this(Color.Green, 0.0f, 0.0f, 181, 181)
    {
        // Using 181 (100 + 81, where 81 is the last two digits based on
the original Shape constructor)
    }

    // Overloaded constructor
    public MyRectangle(Color color, float x, float y, int width, int
height) : base(color)
    {
        X = x;
        Y = y;
        _width = width;
        _height = height;
    }

    // Properties
    public int Width
    {
        get { return _width; }
        set { _width = value; }
    }

    public int Height
    {
        get { return _height; }
        set { _height = value; }
    }
```

```
    // Override Draw method
    public override void Draw()
    {
        if (Selected)
        {
            DrawOutline();
        }
        SplashKit.FillRectangle(Color, X, Y, _width, _height);
    }

    // Override DrawOutline method
    public override void DrawOutline()
    {
        // The outline is 6 pixels wider on all sides
        SplashKit.DrawRectangle(Color.Black, X - 6, Y - 6, _width + 12,
_height + 12);
    }

    // Override IsAt method
    public override bool IsAt(Point2D pt)
    {
        return pt.X >= X && pt.X <= (X + _width) &&
               pt.Y >= Y && pt.Y <= (Y + _height);
    }

    // Step 6: Override SaveTo method
    public override void SaveTo(StreamWriter writer)
    {
        writer.WriteLine("Rectangle");
        base.SaveTo(writer);
        writer.WriteLine(Width);
        writer.WriteLine(Height);
    }

    // Step 14: Override LoadFrom method
    public override void LoadFrom(StreamReader reader)
    {
        base.LoadFrom(reader);
        Width = reader.ReadInteger();
        Height = reader.ReadInteger();
    }
}
```

MyCircle.cs

```
using System;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer;

public class MyCircle : Shape
{
    // Fields
    private int _radius;

    // Default constructor
    public MyCircle() : this(Color.Blue, 0.0f, 0.0f, 131)
    {
        // Using 131 (50 + 81, where 81 is the last two digits)
    }

    // Overloaded constructor
    public MyCircle(Color color, float x, float y, int radius) :
base(color)
    {
        X = x;
        Y = y;
        _radius = radius;
    }

    // Properties
    public int Radius
    {
        get { return _radius; }
        set { _radius = value; }
    }

    // Override Draw method
    public override void Draw()
    {
        if (Selected)
        {
            DrawOutline();
        }
        SplashKit.FillCircle(Color, X, Y, _radius);
```

```
    }

    // Override DrawOutline method
    public override void DrawOutline()
    {
        // Draw a black circle with a radius 2 pixels larger
        SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
    }

    // Override IsAt method
    public override bool IsAt(Point2D pt)
    {
        // Check if the point is within the circle using SplashKit's
helper method
        return SplashKit.PointInCircle(pt, SplashKit.CircleAt(X, Y,
_radius));
    }

    // Step 7: Override SaveTo method
    public override void SaveTo(StreamWriter writer)
    {
        writer.WriteLine("Circle");
        base.SaveTo(writer);
        writer.WriteLine(Radius);
    }

    // Step 15: Override LoadFrom method
    public override void LoadFrom(StreamReader reader)
    {
        base.LoadFrom(reader);
        Radius = reader.ReadInteger();
    }
}
```

MyLine.cs

```
using System;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer;

public class MyLine : Shape
{
    // Fields
    private float _endX;
    private float _endY;

    // Default constructor
    public MyLine() : this(Color.Red, 0.0f, 0.0f, 100.0f, 100.0f)
    {
    }
    // Overloaded constructor
    public MyLine(Color color, float startX, float startY, float endX,
float endY) : base(color)
    {
        X = startX;
        Y = startY;
        _endX = endX;
        _endY = endY;
    }

    // Properties
    public float EndX
    {
        get { return _endX; }
        set { _endX = value; }
    }

    public float EndY
    {
        get { return _endY; }
        set { _endY = value; }
    }

    // Override Draw method
    public override void Draw()
```

```
    {
        if (Selected)
        {
            DrawOutline();
        }
        SplashKit.DrawLine(Color, X, Y, _endX, _endY);
    }

    // Override DrawOutline method
    public override void DrawOutline()
    {
        // Draw small circles around the start and end points
        SplashKit.FillCircle(Color.Black, X, Y, 5);
        SplashKit.FillCircle(Color.Black, _endX, _endY, 5);
    }

    // Override IsAt method
    public override bool IsAt(Point2D pt)
    {
        // Check if the point is on the line using SplashKit's helper
method
        Line line = SplashKit.LineFrom(X, Y, _endX, _endY);
        return SplashKit.PointOnLine(pt, line, 5.0f);
    }

    // Step 8: Override SaveTo method
    public override void SaveTo(StreamWriter writer)
    {
        writer.WriteLine("Line");
        base.SaveTo(writer);
        writer.WriteLine(EndX);
        writer.WriteLine(EndY);
    }

    // Step 15: Override LoadFrom method
    public override void LoadFrom(StreamReader reader)
    {
        base.LoadFrom(reader);
        EndX = reader.ReadSingle();
        EndY = reader.ReadSingle();
    }
}
```

TextDrawing.txt

1
1
1
6
Circle
0
0
1
425
230
131
Rectangle
0
0.49803922
0
81
412
181
181
Rectangle
0
0.49803922
0
563
38
181
181
Line
1
0
0
446
443
546
443
Line
1
0
0
634
487

734
487
Circle
0
0
1
133
134
131